University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 2: this Friday @ 11:59 pm
  - Lab 1: next Monday @ 11:59 pm
  - Assignment 1: Monday Oct 10th @ 11:59 pm
- TAs student support hours available on the syllabus page

Previous lecture

- BinaryTree implementation
  - privateBuildTree method
  - traversals

Muddiest Points

- **Q: Arguments vs Parameters - when/is it a shallow copy or a deep copy, and how can the parameters affect the arguments**
- **Q: Can you go over the difference again between left tree/right tree and tree A and tree B**
- A: When a method is called, arguments are copied into parameters. It is always a shallow copy. Parameters affect arguments when they are used to invoke mutating methods on the objects.
- **Q: Why does leftTree= null; and rightTree=null; not block access from clients?**
- A: Assigning to parameters has no effect on the arguments, which would still have access to the internal nodes of the tree
- **Q: how using clear() prevents client access to nodes**
- A: Because clear() is a mutating method that sets the root instance variable to null.

Muddiest Points

- **Q: I think the purpose of the tree builder method is to combine two trees? By assigning their roots to the left and right children of the new root of the new tree?**
- A: Correct! This bottom-up approach is by the way how a binary tree is typically built. You start from the leaves, combine them to form a bigger subtree, combine subtrees to form larger subtrees, …
- special case where treeA == treeB
- **Q: Are there any other ways to reliably prevent client access to the left and right trees without setting their roots to null?**
- A: Another option was suggested in class (by Sahas?) to make a deep copy of the left and right trees

Muddiest Points

- **Q: are there non-deep copies?**
- A: Yes. root = other.root; is a shallow copy
- **Q: Deep vs. deeper copy. I originally thought that the <T> type was the same thing as a binary node, but then it made sense that the binary node contained a pointer to each <T>. I think I got it now.**
- A: Yes, BinaryNode objects contain references to T objects
- **Q: can you assume that the copy() function is shallow? how can you tell if it is a deep or shallow copy besides the fact that the data is also copied?**
- A: copy() is not a standard Java method. You will have to depend on the documentation to know if it does shallow, deep, or deeper copy
- **Q: what purpose does a deep but not deeper copy serve?**
- A: If T is immutable, a deep copy is the same as a deeper copy. If T is mutable, a deep copy can be more efficient as it allows for *lazy copy*, that is, duplicating an object only upon attempting to modify it

Muddiest Points

- **Q: If you want to process nodes going up the binary tree with a recursive call, do you put the operations before or after the recursive call?**
- A: After the recursive call

Muddiest Points

- **Q: the cases where leftTree/rightTree/treeA/treeA are equal to this…are they references to this? Why is this a problem?**
- A: This special case is when they *== this*, not just *.equals(this).* It is a problem because we are calling leftTree.clear() and rightTree.clear() to prevent client access. We don't want to clear this after building it!

Muddiest Points

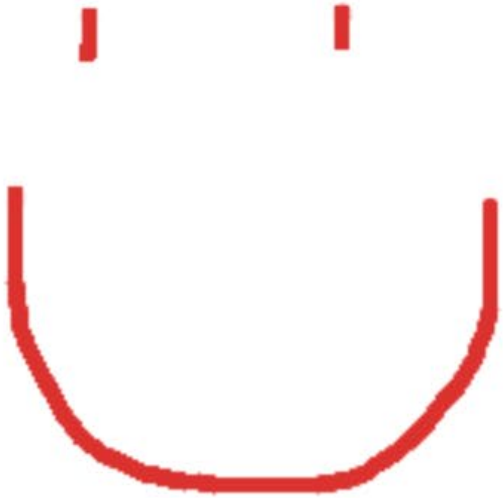
- **Q: what is the different traverses for the binary trees? Can you show each traverse type with an example.**
- A: pre-order, post-order, in-order, and level-order
- **Q: what type of traversal you use for what function?**
- A:
  - Pre-order → prefix (Polish) notation of an expression parse tree
  - In-order → retrieve nodes of a Binary Search Tree in sorted order
  - Post-order → Compute a property of tree nodes based on properties of node's children

Muddiest Points

- **Q: what are implementations of binary tree in daily life**
- A:
  - Data compression (Hoffman tree)
  - Databases (B-Tree)
  - Machine learning applications (Decision tree)
  - Internet routing (Trie)

Muddiest Points

Anonymous



:)

Comments 0 1

This Lecture

- Binary Search Tree
  - How to add and delete
- Runtime of BST operations
  - Find, add, delete
- Red-Black BST (Balanced BST)
  - definition and basic operations

Binary Search Tree

- Search Tree Property
  - For each node in the tree:
    - The data of the node is larger than the data in all nodes in the node's left subtree and
    - The data of the node is smaller than the data in all nodes in the node's right subtree

Let's build a Binary Search Tree

- Work in groups of 2-3 students
- Add the following integers to a Binary Search Tree in **the following order**:

10, 8, 17, 7, 5, 20, 15, 16, 4

Reflect on the steps that you followed

- How did you add 4 to the tree?
- What steps did you follow?

10, 8, 17, 7, 5, 20, 15, 16, 4

BST: How to add?

- How to add a data item *entry* into a BST rooted at *root?*
- What if root.data.compareTo(entry) == 0?
- What if root.data.compareTo(entry) < 0?
  - Move left or right?
  - What if no child?
  - What if there is a child?
- What if root.data.compareTo(entry) > 0?
  - Move left or right?
  - What if no child?
  - What if there is a child?
- What if I tell you that you have a friend who can add into a BST.
  - How can you use the help of that friend?

Let's see the code for adding into a BST

- Available online at:
  - https://cs1501-2231.github.io/slides-handouts/CodeHandouts/TreeADT/Slides
  - The slides are under the CodeHandouts/TreeADT/slides folder in the handout repository
  - https://github.com/cs1501-2231/slides-handouts

Let's build a Binary Search Tree

- Work in groups of 2-3 students
- Add the following integers to a Binary Search Tree in **the following order**:

4, 5, 7, 8, 10, 8, 15, 16, 17, 20

Reflect on the steps that you followed

- How many comparisons did you have to make to add 20?

4, 5, 7, 8, 10, 8, 15, 16, 17, 20

Run-time of BST operations

- For add, # comparisons = d, where *d* is the depth of the new node
- For search miss, # comparisons = d, where *d* is the depth of the node if it were in the tree
- For search hit, # comparisons = 1+d, where *d* is the depth of the found node
- On average, node depth in a BST is O(log n)
  - *n* is the number of data items
  - Proof in Proposition C in Section 3.2 of Sedgewick Textbook
- In the worst case, node depth in a BST is O(n)

Let's switch to delete!

- Work in groups of 2-3 students
- In the Binary Search Tree that you built out of **the following order**:

10, 8, 17, 7, 5, 20, 15, 16, 4

- How would you delete 4?
- How would you delete 5?
- How would you delete 10?

BST: How to delete?

- Three cases
  - Case 1: node to be deleted is a leaf
    - Easiest case!
    - Pass back null to the node's parent
  - Case 2: node to be deleted has one child
    - Pass back the child to the node's parent to adopt instead of the node

BST: How to delete?

- Three cases
  - Case 3: node to be deleted has two children
    - Difficult to delete the node!
      - It has two children, so parent cannot adopt both
    - Let's try to replace the data in the node
      - We still want to maintain the search tree property
      - What are our options?
    - Replace node's data by its successor
      - largest item in left subtree
    - or by its predecessor
      - smallest item in right subtree
    - Delete the node that we selected in the previous step
      - Has to have at most one child
      - Why?

Let's see the code for deleting from a BST

- Available online at:
  - https://cs1501-2231.github.io/slides-handouts/CodeHandouts/TreeADT/Slides
  - The slides are under the CodeHandouts/TreeADT/slides folder in the handout repository
  - https://github.com/cs1501-2231/slides-handouts